بسمه تعالی

تمرین سری یازدهم درس یادگیری عمیق

پوریا محمدی نسب

۴۰۰۷۲۲۱۳۸

سوال ۱.

در این تمرین میخواهیم یک سیستم پرسش و پاسخ را با استفاده از مجموعه داده این لینک طراحی کنیم. توضیحات هر بخش و نحوه استفاده از داده ها و مدل ها در نوتبوک تمرین آورده شده است. لازم است قسمت های مشخص شده را مطالعه و کامل نمایید. سپس نتایج به دست آمده را به طور دقیق گزارش و تحلیل نمایید. همچنین مزایا و معایب مدل پیشنهادی را ذکر نمایید و بفرمایید چگونه در کارهای واقعی میتوان از آن استفاده نمود.(این سوال برای آمادگی بیشتر برای درک بهتر مدل های مبدل می باشد و ۱۵ نمره ی امتیازی دارد.)

داخل کد پیاده سازی شده (ضمیمه شده همراه با فایل گزارش) دو سری از تسک های QA1 و QA2 را انجام داده ایم که از دو شبکه عصبی بازگشتی (LSTM-Based) استفاده کرده ایم که کار این دو مدل این است که بر اساس Story و Question آموزش ببینند. در ادامه یک بردار ادغام شده داشته باشیم که به عنوان Query در نظر گرفته میشود تا بتوانیم جوابهای مناسب برای سوالهای موجود در تسک های متفاوت ارائه دهیم. نتایج زیر خروجی ما خواهد بود:

Model	Accuracy	
	Task QA1	Task QA2
GRU	100	50
LSTM Base	50	20

واضح است که GRU در این سوال از LSTM بهتر عمل میکندو نتایج همچنین نشان میدهند که هر چه طول سوال بلند تر باشد و نویز به آنها اضافه کنیم دقت ها کاهش میابند. اگر از attention استفاده کنیم ممکن است بتوانیم در نویزها نیز جستجو کنیم ئ موارد مرتبط را کشف کنیم که باعث بهبود عملکرد مدل خواهد شد.

اخیرا تلاش های بسیاری صورت گرفته است تا ساختار عبارات بلند در دنباله ها را با استفاده از RNN و LSTM شناسایی کنیم. حافظه در این مدل ها حالت grid دارد که در بازه های زمانی طولانی نهفته و ناپایدار است. مدل های مبتنی بر LSTM این موضوع را از طریق سلول های حافظه محلی در حالت شبکه از گذشته قفل میشوند برطرف میکنند. در عمل، افزایش عملکرد نسبت به RNN های آموزش دیده با دقت اندک است. .مدل ما با اینها تفاوت دارد زیرا از یک حافظه سراسری، با توابع خواندن و نوشتن مشترک استفاده میکند. با این حال، با وزن های مبتنی بر لایه ها، مدل ما را میتوان به عنوان شکلی از RNN مشاهده کرد که تنها پس از تعداد گام های زمانی ثابت خروجی تولید میکند و مراحل میانی شامل ورودی حافظه است. به این منظور که مدل های LSTM به خوبی عمل کند ولی در تسک های طولانی تر ضعیف تر عمل میکند زیرا بدست آوردن وابستگی با فاصله ی زیاد میان کلمات کار دشواری است. برای استفاده در عمل یا باید مدل را جایگزین کنیم و از مدل های کمی بهتر استفاده کنیم و یا اینکه بر روی همین مدل، مقداری تغییر را داشته باشیم و این تغییرات میتوانند مانند تغییراتی باشند که در مقاله رفرنس [10] به آن اشاره شده است.

سوال ۲.

چگونه میتوانیم بازده محاسباتی و حافظه را در مدلهای مبتنی بر مبدل بهبود بخشیم؟ (راهنمایی: به بخش مبدل ها در کتاب و مقاله [Tay et al. 2020] مراجعه کنید.)

هزینه محاسباتی transformer ها با توجه به عوامل متفاوتی قابل محاسبه است. در اولین گام، حافظه و پیچیدگی محاسباتی مورد نیاز برای محاسبه ماتریس attention در طول توالی ورودی، درجه دو است(N^2). عملیات ضرب ماتریس به تنهایی زمان و حافظه ی N^2 مصرف میکند. در این حالت در مدل های self-attention در برنامه هایی که نیاز به پردازش دنباله های طولانی دارند، محدود میشوند. بیشترین محدودیت حافظه در موقع آموزش مدل هاست زیرا نیاز است تا گرادیان وزن ها محاسبه و بروزرسانی شود. هزینه درجه دوم self-attention نیز بر سرعت آموزش و استنتاج نهایی تاثیر میگذارد. هزینه های محاسباتی مکانیزم self-attention تا حدی بر هزینه محاسباتی کلی مبدل کمک میکند. علت این سربار محاسباتی به خاطر وجود لایه های FFN در هر بلوک است. پیچیدگی FFN با توجه به طول دنباله خطی است اما به طور کلی هنوز پر هزینه است. توجه کارآمد و مدل های کارآمد عموما متعامد هستند. البته برخی از روش های attention کارآمد به صراحت با هدف کاهش طول دنباله و صرفه جویی در هزینه محاسباتی زمانی و حافظه ای معرفی شده اند. از راه های کاهش هزینه محاسباتی میتوان به محدود کردن Field of view ، استفاده از یک ماژول حافظه جانبی و کاهش رزولوشن اطلاعات ورودی اشاره کرد. در ادامه یک طبقه بندی کلی از مدل های transformer را نشان میدهیم که هدف اصلی بیشتر این مدل ها بهبود پیچیدگی حافظه در مکانیزم self-attention است. روش های زیر کارایی کلی معماری مبدل را بهبود میبخشند.

- Fixed Patterns: اولین تغییرات در self-attention به سادگی در ماتریس توجه با محدود کردن قسمت دید به الگوهای پیشفرض مانند پنجره های محلی را پراکنده میکند. الگوهای block-wise ساده ترین مثال از این تکنیک است که بلوک قسمت گیرنده محلی را با تقسیم کردن دنباله های ورودی XOR در نظر میگیرد. تقسیم توالی ورودی به بلوک ها، پیچیدگی ها را از N^2 به B^2 به صورتی که B بسیار کوچک تر از N باشد کاهش میدهد و هزینه به طور قابل توجهی کاهش میابد. این روش های block-wise به عنوان پایه ای برای بسیاری از مدل های پیچیده تر عمل میکنند.
- Combination of Patterns: ایده کلیدی رویکردهای ترکیبی، بهبود پوشش با ترکیب دو با چند الگوری دسترسی متمایز است. به عنوان مثال، Sparse Transformer با اختصاص نیمی از سرهای خود به هر الگو، توجه گام به گام و محلی را ترکیب میکند. به طور مشابه، Axial Transformer دنباله ای از محاسبات self-attention را اعمال میکند که یک tensor با ابعاد بالا به عنوان ورودی اعمال میکند. در واقع ترکیب الگوها پیچیدگی حافظه را به همان روش fixed patterns کاهش میدهد. با این حال تفاوت بین این دو روش این است که تجمع و ترکیب الگوهای متعدد پوشش کلی مکانیزم self-attention را بهبود میبخشد.
- Performer: مدل Performer با مکانیزم توجه تعمیم یافته و از هسته‌های تصادفی استفاده می‌کند. پیچیدگی پارامتر، حافظه و هزینه محاسباتی آن O(Md+Nd+MN) است که در آن M ابعاد ویژگی‌های تصادفی است. شایان ذکر است که تغییرات یک جهته را نمی‌توان به‌صورت یک جهته و با زمان خطی پنهان کرد. این امر باعث می‌شود که آموزش مجری در یک کار مولد بسیار کند شود. زمان استنتاج برای رمزگشایی افزایشی، از افزایش سرعت سودمند نخواهد بود.
- Synthesizers: تلاشی برای مطالعه و بررسی اهمیت واقعی شرطی‌سازی در مکانیزم توجه به خود هستند و همچنین اولین تلاش‌ه برای اختلاط نشانه‌های بدون قید و شرط است. Synthesizerها تنها به‌طور ضمنی با مبدل‌های کارآمد

مرتبط هستند و می‌توانند بیشتر به‌عنوان یک MLP-Mixer درنظر گرفته شوند. برای Synthesizer قابل‌آموزش، پیچیدگی حافظه و پیچیدگی پارامتر N^2 باقی می‌ماند. البته هزینه‌ی محاسباتی به میزان قابل توجهی کاهش می‌یابد. همچنین این مدل هزینه‌های پارامتر را به $2(N*K)$ کاهش می‌دهد.

سوال ۳.

در این قسمت میخواهیم یک مدل Transformer پیاده سازی کنیم. سه بخش مهم به همراه توضیحات لازم در نوتبوک تمرین آورده شده است. ابتدا آنها را مطالعه نمایید و در قسمت های مشخص شده پیاده سازی های لازم را انجام دهید و به سوالات زیر پاسخ نیز دهید:

الف)

چه زمانی token type ids مفید است؟

در مواردی هدف مدل transformer رده‌بندی (classification) روی جفت های پرسش و پاسخ است. در این حالت نیاز است تا ۲ دنباله متفاوت به صورت یک ورودی input_id با هم الحاق شوند که معمولا با استفاده از توکن های مخصوص این کار صورت میگیرد.(مانند توکن های [CLS] و [SEP]). برای مثال در BERT برای انجام این کار به صورت زیر عمل میکنیم:

[CLS] Sequence A [SEP] Sequence B [SEP]

حال اگر قصد داشته باشیم که از tokenizer خود به گونه ای استفاده کنیم که به صورت خودکار ۲ دنباله ورودی را دریافت کند و توکن های مخصوص را در آن قرار دهد میتوان صرفا با مشخص کردن ابتدا و انتهای دنباله ها توکن مناسب در آنها قرار داد. اما در مدل هایی مانند BERT از token type IDs که با Segment IDs هم شناخته میشوند به کل دنباله های ورودی یک ماسک دودویی اختصاص میدهد. به طوری که برای توکن های مربوط به Question مقدار 0 و به توکن های Answer مقدار 1 قرار داده میشود. Segment IDs در مواردی که بیش از یک sequence/part در تنسور token IDs وجود دارد، مفید هستند.

ب)

ایده اصلی مدل های transformer چیست؟

مدل های transformer معماری های جدیدی در زمینه NLP هستند که اکثرا برای کاربرد های وظایف تبدیل یک دنباله به دنباله دیگر و کارکردن با دنباله هایی با وابستگی طولانی مورد استفاده قرار میگیرند. برای اولین باید ایده استفاده از transformer ها به عنوان یک مدل زبانی در مقاله رفرنس [3] بیان شد. همانطور که قبلا نیز ذکر شد یکی از مهم ترین کاربردهای transformer ها تبدیل دنباله ورودی به دنباله خروجی است. ایده اصلی مدل های transformer استفاده از attention و بازگشتی بودن کامل است.

ج)

اگر توالی های ورودی بسیار طولانی باشند، چه چالش هایی برای مبدل ها وجود دارد؟ چرا؟

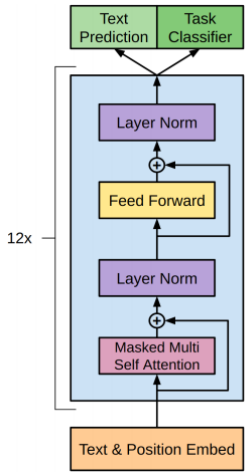
بیشتر مدل های transformer طول دنباله را ثابت میگیرند. مانند مدل معروف BERT که طول توکن را 512 در نظر میگیرد. این تعداد برای طیف وسیعی از مسائل کافی است و مشکلی وجود ندارد اما وقتی روی تسکی مانند document summarization و Question answering کار میکنیم میتواند باعث مشکلات جدی شود. در این تسک ها هندل کردن یک محتوای بسیار طولانی مهم است. اما در عین حال ممکن از حداکثر طولی که مدل برای توکن ها در نظر گرفته است برای حفظ ارتباط اطلاعات قبلی کافی نباشد. دلیلی که transformer ها دارای محدودیت در طول توکن ها هستند هزینه

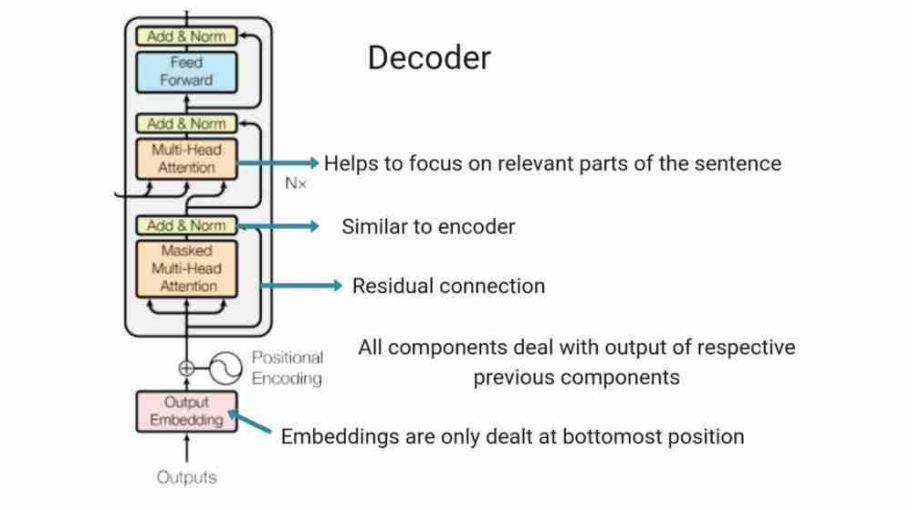
محاسباتی و پیپیدگی حافظه است که میتواند باعث شود آموزش مدل عملا قابل انجام نباشد. برای رفع این مشکل یک راه حل استفاده از یک ارتباط خطی در حفظ کردن اطلاعات قبلی است که به efficient transform معروف است.

سوال ۴.

برای مدل سازی زبانی از مبدل های رمزنگار (encoder)، رمزگشا (decoder) یا هر دو استفاده میشود؟ چگونه این روش را طراحی کنیم؟ (راهنمایی: برگرفته از سوالات بخش مبدل ها در کتاب)

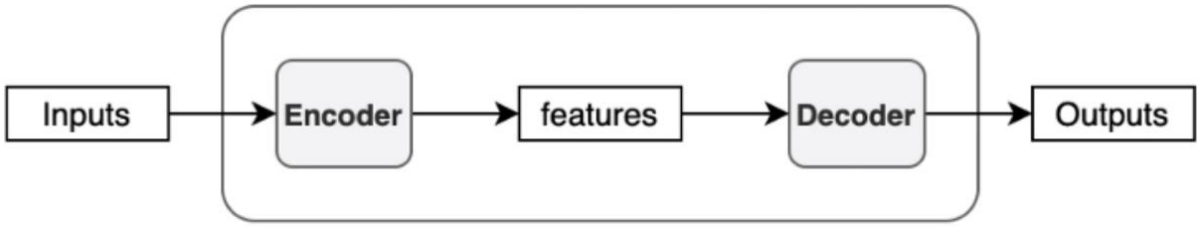
نیاز به encoder بستگی به کاربردی دارد که به دنبال آن هستیم، به عنوان مثال : در مدل‌های زبانی علّی (سنتی) (LMs)، هر توکن به شرط توکن های قبلی پیش بینی می شود. با توجه به اینکه توکن های قبلی توسط خود decoder دریافت می شوند، نیازی به encoder نداریم. در مدل‌های ترجمه ماشین (NMT)، هر توکن با استفاده از توکن های ترجمه شده پیشین و متن اصلی ترجمه می شود. توکن های قبلی توسط decoder دریافت می شوند، اما متن اصلی توسط encoder پردازش می شود. در LM های ماسک دار، مانند BERT، پیش‌بینی توکن ماسک دار به بقیه توکن های جمله مشروط می‌شود. این ها در encoder دریافت می شوند، بنابراین نیازی به decoder نداریم. همچنین به این صورت نیز می توان گفت که معماری encoder-decoder برای زمانی است که می خواهیم دو متن متفاوت را به هم تبدیل کنیم مثلا یک متن از زبان انگلیسی را به زبان فرانسوی ترجمه کنیم در این حالت با توجه به متفاوت بودن دو متن نیاز است تا به هرکدام از آن ها وزن متفاوتی بدهیم در نتیجه نیاز به هر دو بخش decoder و encoder داریم.اما اگر ورودی ما یک متن باشد که تنها بخواهیم ادامه آن را تولید و پیش بینی کنیم چون کل ورودی از یک جنس می باشد نیازی به وزن های متفاوت نیست که در این حالت استفاده از بخش decoder به تنهایی کفایت می کند. در این حالت لایه attention فقط به کلمات قبلی دنباله دسترسی دارد. در این نوع معماری ما به دنبال پیش بینی کردن کلمه بعدی دنباله هستیم در نتیجه عموما این مدل برای text generation به کار می رود. مثالی از این نوع معماری GPT-2 می باشد. به عنوان مثالی از معماری که تنها decoder داشته باشد نیز می توان Bert را نام برد. از Bert می توان در تولید مدل زبانی استفاده کرد که چون در این حالت نیاز به کل دنباله داریم تا بتوانیم احتمالات کلمات مختلف را به دست بیاوریم پس تنها با یک decoder می توان مدل را طراحی کرد.



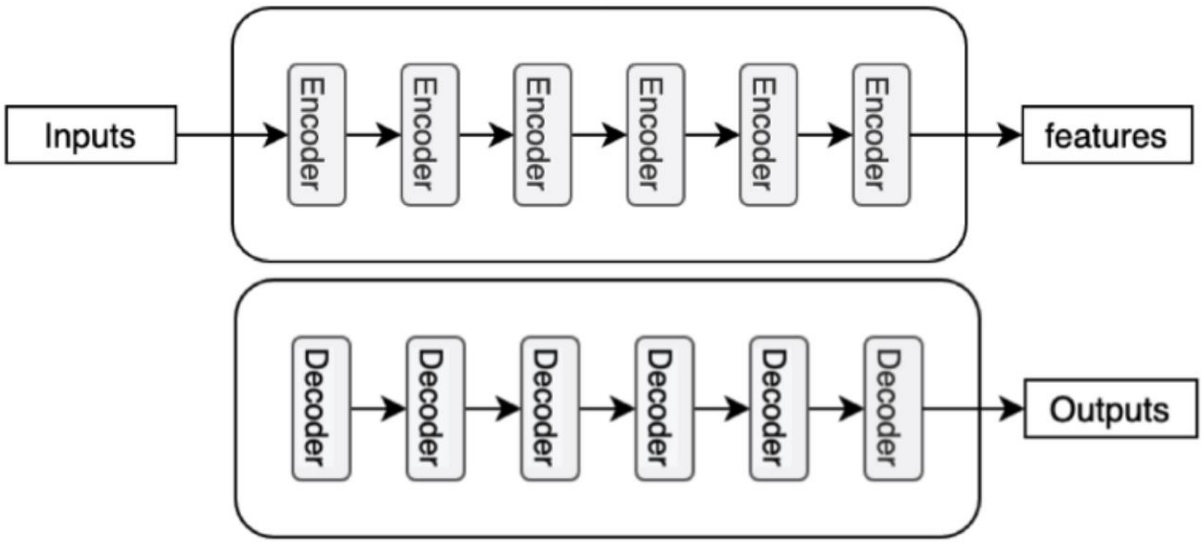


اما به نظر میرسد بهترین مدل هم از encoder و هم از decoder استفاده میکند. معماری آن را به صورت مفصل شرح میدهیم. در مقاله اصلی transformer ها [7]، این مدل ها برای ترجمه ماشینی معرفی شدند. به طور مثال سعی داریم مدلی آموزش دهیم که جمله ای از یک زبان را به زبان دیگر ترجمه کند.

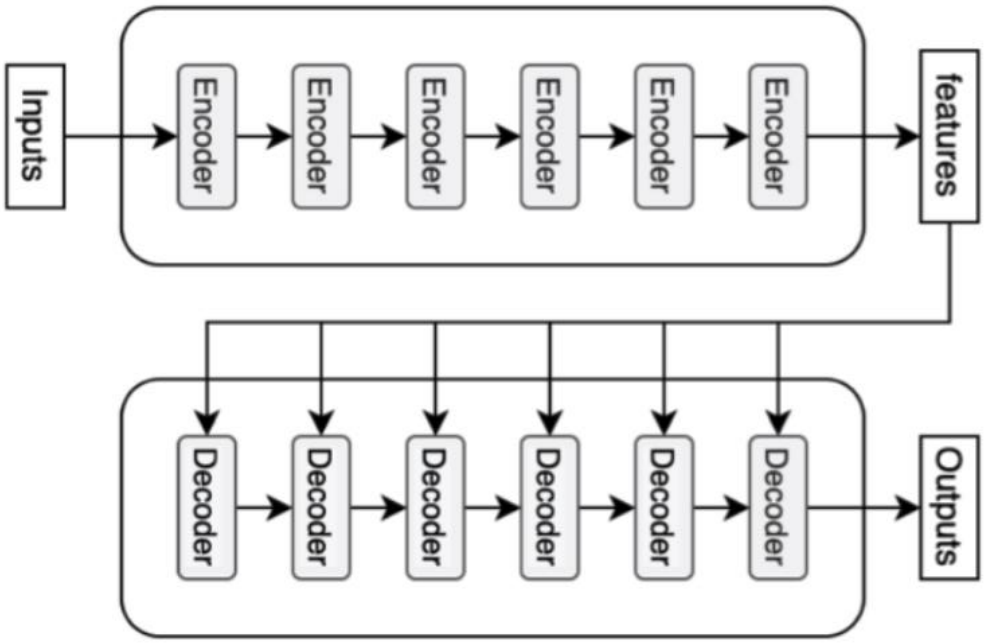
حال به داخل transformer مناسب برای مدل های زبانی نگاهی بیاندازیم:



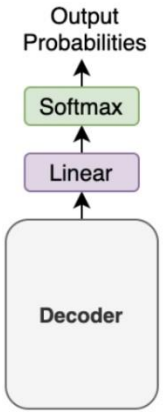
همانطور که درتصویر بالا مشاهده میشود دنباله ورودی با استفاده از encoder تبدیل به یک بردار ویژگی قابل بازنمایی (Representative) میشود. و این بردار ویژگی به عنوان ورودی decoder باعث میشود تا شبکه به دنباله مد نظر خود برسد. نکته قابل توجه این است که داخل قسمت encoder شکل بالا در واقع چندین بلوک encoder تعبیه شده است . همین ویژگی برای decoder نیز صادق است. شکل زیر مفهوم جملات بالا را بیان میکند.



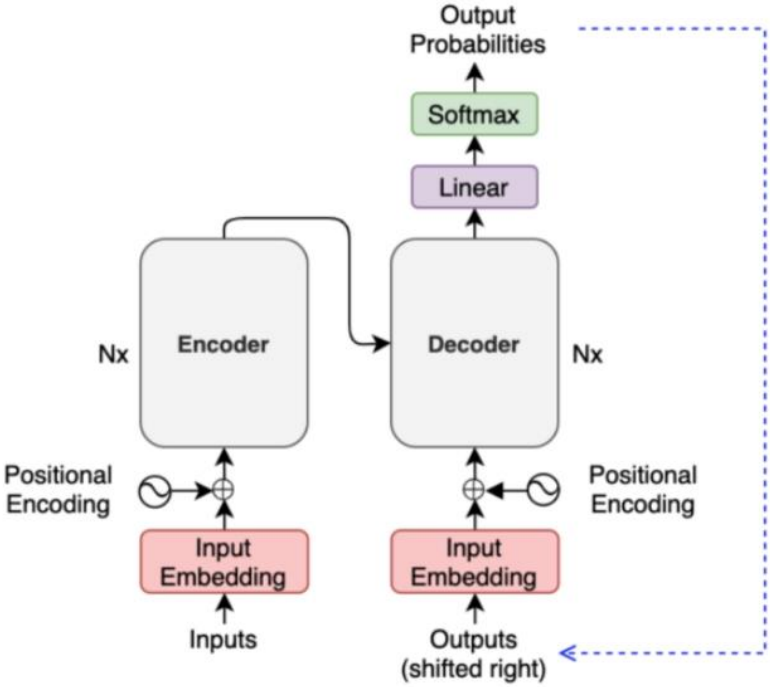
هر بلوک decoder بردارهای ویژگی خارج شده از encoder متناظر را دریافت میکند و شکل کلی شبکه transformer به صورت زیر است.



همچنین برای آموزش مدل زبانی، از تابع فعالسازی softmax در خروجی decoder استفاده میشود. خروجی decoder یک توکن در هر واحد زمانی است. خروجی decoder به عنوان قسمتی از دنباله ورودی مرحله بعد استفاده میشود که به این روش استفاده از خروجی به عنوان ورودی مرحله بعد auto-regressive میگویند که به مدل اجازه میدهد دنباله خروجی را در طول های مختلفی نسبت به طول دنباله ورودی تولید کند.

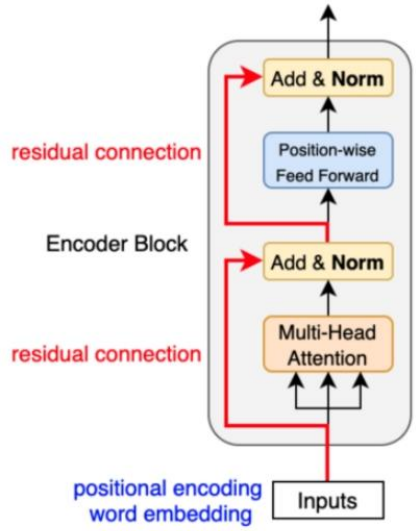


پس اگر اتصال خروجی decoder به ورودی مرحله بعد آن را در نظر بگیریم مدل transformer ما برای مدل زبانی به شکل زیر است:

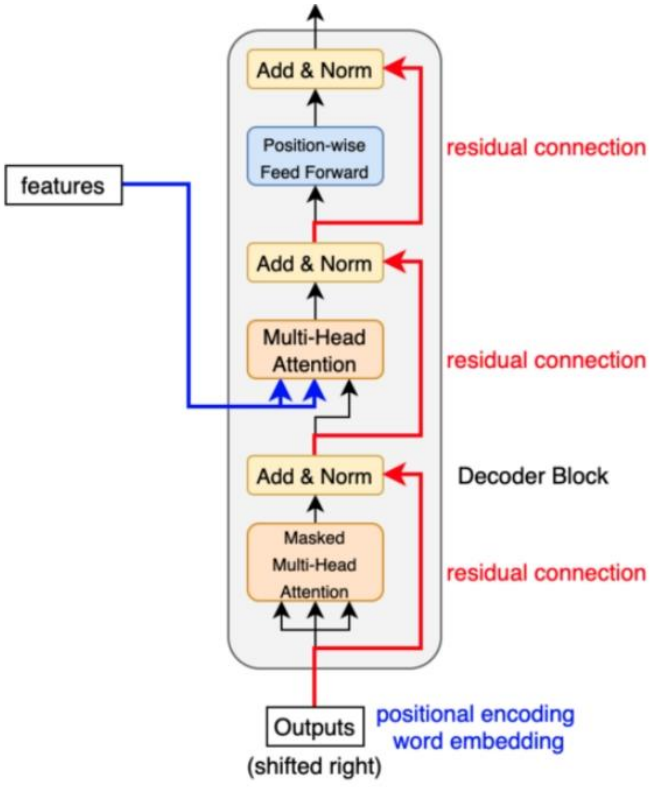


حال کمی با جزئیات بیشتری به ساختار دارخل بلوک های encoder و decoder میپردازیم.

بلوک encoder از یک مکانیزم self-attention استفاده میکند تا هر توکن را با استفاده از اطلاعاتی که در محتوای جمله است غنی سازی کند. هر توکن ممکن است چند معنی یا عملکرد داشته باشد پس از توجه چند سر استفاده میشود تا این مورد را حل کند. همینطور قسمتی تحت عنوان FFN در encoder وجود دارد که یک لایه خطی با تابع فعالسازی ReLU و یک لایه خطی دیگر است که هر بردار embedding را به صورت جدا جدا پردازش میکند. همچنین در ساختار بلوک encoder از ایده residual block نیز استفاده میشود که هر residual connection در واقع embedding قبلی را تا ایه ی زیر دنباله حمل میکند. همچنین در مقاله اصلی transformer بیان شده است که استفاده از Layernormalization مفید است. پس هر واحد بلوکی encoder به طور کامل به شکل زیر خواهد بود.



به طور مشابه به بررسی ساختار دقیق داخل بلوک decoder میپردازیم. معماری داخل بلوک decoder بسیار شبیه به معماری encoder است با این تفاوت که در decoder توجه source-target نیز محاسبه میشود.



مراجع

1) https://huggingface.co/docs/transformers/glossary

2) towardsdatascience.com/why-are-there-so-many-tokenization-methods-for-transformers

3) Al-Rfou, Rami, et al. "Character-level language modeling with deeper self-attention." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. No. 01. 2019.

4) https://www.analyticsvidhya.com/understanding-transformers-nlp-state-of-the-art-models/

5) https://towardsdatascience.com/transformers-141e32e69591

6) https://medium.com/@lukas.noebauer/the-big-picture-transformers-for-long-sequences

7) Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

8) https://datascience.stackexchange.com/why-is-the-decoder-not-a-part-of-bert-architecture

9) https://www.kaggle.com/general/236840

10) Sukhbaatar, Sainbayar, Jason Weston, and Rob Fergus. "End-to-end memory networks." *Advances in neural information processing systems* 28 (2015).

11) https://github.com/comet-ml/comet-examples/blob/mastercomet-keras-bAbI-example.py